
Lecture 18: More Dataflow Analysis

COS 320

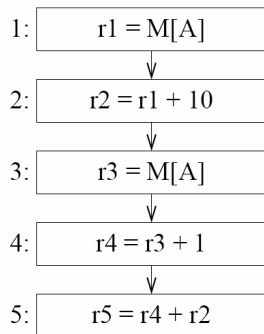
Compiling Techniques

Princeton University
Spring 2007

Prof. David August

Common Subexpression Elimination

If $x \text{ op } y$ is computed multiple times, *common subexpression elimination* (CSE) attempts to eliminate some of the duplicate computations.



Need to track expression propagation → available expression analysis

Definitions

- Expression $x \text{ op } y$ is *available* at CFG node n if, on every path from CFG entry node to n , $x \text{ op } y$ is computed at least once, and neither x nor y are defined since last occurrence of $x \text{ op } y$ on path.
- Can compute set of expressions available at each statement using system of dataflow equations.
- Statement $r1 = M[r2]$:
 - *generates* expression $M[r2]$.
 - *kills* all expressions containing $r1$.
- Statement $r1 = r2 + r3$:
 - *generates* expression $r2 + r3$.
 - *kills* all expressions containing $r1$.

Iterative Dataflow Analysis Framework

- Specify:
 - Two set definitions - $A[n]$ and $B[n]$
 - A transfer function - $f(A, B, IN/OUT)$
 - A confluence operator - \vee .
 - A direction - FORWARD or REVERSE.

- For forward analyses:

$$IN[n] = \vee_{p \in PRED[n]} OUT[p]$$

$$OUT[n] = f(A, B)$$

- For reverse analyses:

$$OUT[n] = \vee_{s \in SUCC[n]} IN[s]$$

$$IN[n] = f(A, B)$$

Available Expression Analysis

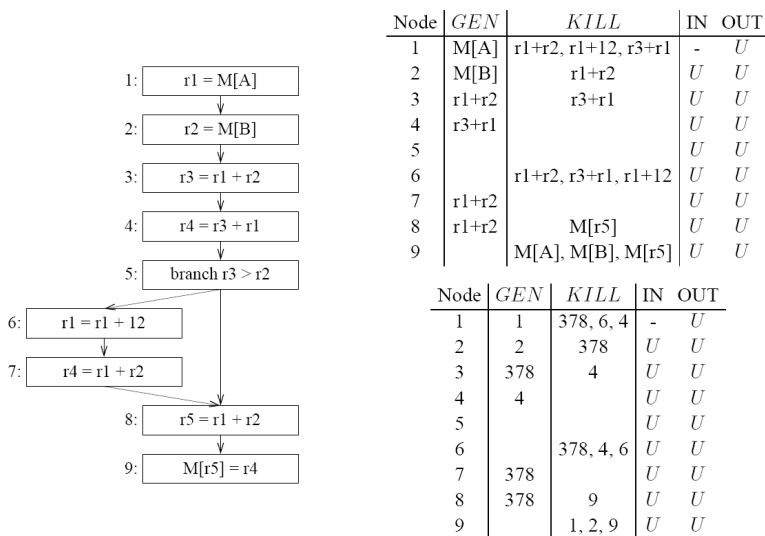
Available Expression Analysis:

- $exp(t)$ - set of all expressions containing t .
- Set definition ($A[n]$): $GEN[n]$ - the set of all expressions generated by n .
- Set definition ($B[n]$): $KILL[n]$ - the set of all expressions that n kills - $exp(n)$.
- Transfer function ($f(A, B, IN/OUT)$): $GEN[n] \cup (IN[n] - KILL[n])$
- Confluence operator (\vee): \cap
 - Use of \cup , required initialization of IN and OUT sets to \emptyset .
 - Use of \cap , requires initialization of IN and OUT sets to U (except for IN of entry node).
- Direction: FORWARD

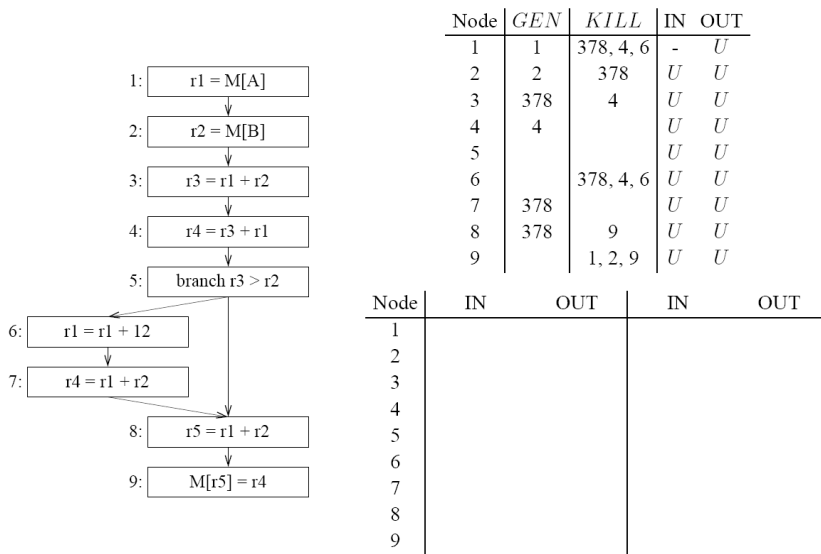
$$IN[n] = \cap_{p \in PRED[n]} OUT[p]$$

$$OUT[n] = GEN[n] \cup (IN[n] - KILL[n])$$

Example



Example



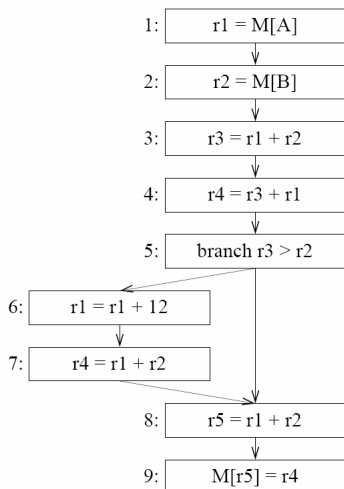
Common Subexpression Elimination (CSE)

Given statement $s: t = x \text{ op } y$:

If expression $x \text{ op } y$ is available at beginning of node s then:

- starting from node s , traverse CFG edges backwards to find last occurrence of $x \text{ op } y$ on each path from entry node to s .
- create new temporary w .
- for each statement $s': v = x \text{ op } y$ found in (1), replace s' by:
 $w = x \text{ op } y$
 $v = w$
- replace statement s by: $t = w$

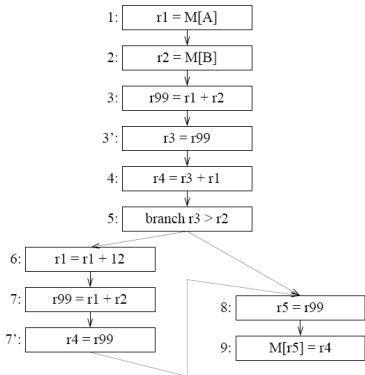
CSE Example



$r1 + r2$ in node 8 is a common subexpression.

Copy Propagation

- Given statement d : $a = z$ (a and z are both register temps) $\rightarrow d$ is a copy statement.
- Given statement u : $t = a \text{ op } b$.
- If d reaches u , no other definition of a reaches u , and no definition of z exists on any path from d to u , then replace u by: $t = z \text{ op } b$.



Sets

- Sets have been used in all the dataflow and control flow analyses presented.
- There are at least 3 representations which can be used:
 - Bit-Arrays:
 - * Each *potential* member is stored in a bit of some array.
 - * Insertion, Member is $O(1)$.
 - * Assuming set size of N and word size of W - Union (OR) and Intersection (AND) is $O(N/W)$.
 - Sorted Lists/Trees:
 - * Each member is stored in a list element.
 - * Insertion, Member, Union, Intersection is $O(size)$. (Insertion, Member is $O(\log_2 size)$ in trees.)
 - * Better for sparse sets than bit-arrays.
 - Hybrids: - Trees with bit-arrays
 - * Use Tree to hold elements containing bit-arrays.
 - * Union, Intersection is $O(size/W)$. Insertion, Member is $O(\log_2 size/W)$.

Basic Block Level Analysis

- To improve performance of dataflow, process at basic block level.
 - Represent the entire basic block by a single *super-instruction* which has any number of destinations and sources.
 - Run dataflow at basic block level.
 - Expand result to the instruction level.
- Example:

```
p: r1 = r2 + r3    ->   r1, r2 = r2, r3
n: r2 = r1
```

Basic Block Level Analysis

- Example:

p: r1 = r2 + r3 -> r1, r2 = r2, r3
n: r2 = r1

- For reaching definitions:

$$OUT[n] = GEN[n] \cup (IN[n] - KILL[n])$$

But $IN[n] = OUT[p]$:

$$OUT[n] = GEN[n] \cup ((GEN[p] \cup (IN[p] - KILL[p])) - KILL[n])$$

Which (clearly) yields:

$$OUT[n] = GEN[n] \cup (GEN[p] - KILL[n]) \cup (IN[p] - (KILL[p] \cup KILL[n]))$$

So:

$$GEN[pn] = GEN[n] \cup (GEN[p] - KILL[n])$$

$$KILL[pn] = KILL[p] \cup KILL[n]$$

- Can we do this at the loop or general region level?

Reducible Flow Graphs Revisited

Definition

- A flow graph is reducible iff each edge exists in exactly one class:
 1. Forward edges (forms an acyclic graph where every node is reachable from start node)
 2. Back edges (head dominates tail)

Algorithm:

1. Remove all backedges
2. Check for cycles:
 - Cycles: Irreducible.
 - No Cycles: Reducible.

Think:

- All loop entry arcs point to header.

Reducible Flow Graphs – Structured Programs

Motivation:

- Structured programs are always reducible programs.
- Reducible programs are not always structured programs.
- Exploit the structured or reducible property in dataflow analysis.

Structures:

- Lists of instructions
- Conditionals/Hammocks
- While Loops (no breaks)

Method:

- Represent structures by a single *super-instruction* which has any number of destinations and sources.
- Run dataflow at structure level.
- Expand result to the instruction level.

Structured Program Analysis

- Lists of instructions - Basic Blocks!

$$GEN[pn] = GEN[n] \cup (GEN[p] - KILL[n])$$

$$KILL[pn] = KILL[p] \cup KILL[n]$$

- Conditionals/Hammocks

$$GEN[lr] = GEN[l] \cup GEN[r]$$

$$KILL[lr] = KILL[l] \cap KILL[r]$$

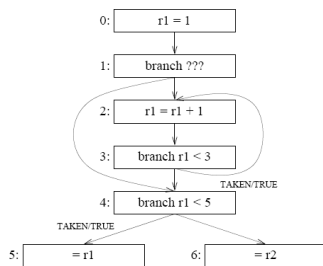
- While Loops

$$GEN[loop] = GEN[l]$$

$$KILL[loop] = KILL[l]$$

Try this on an irreducible flow graph...

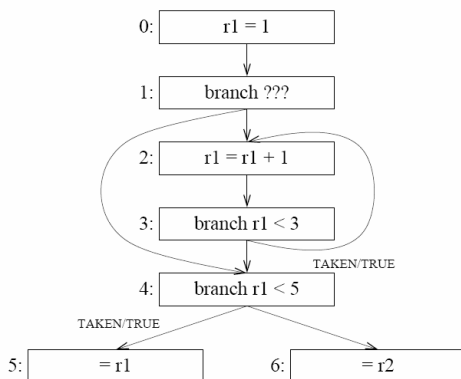
Conservative Approximations



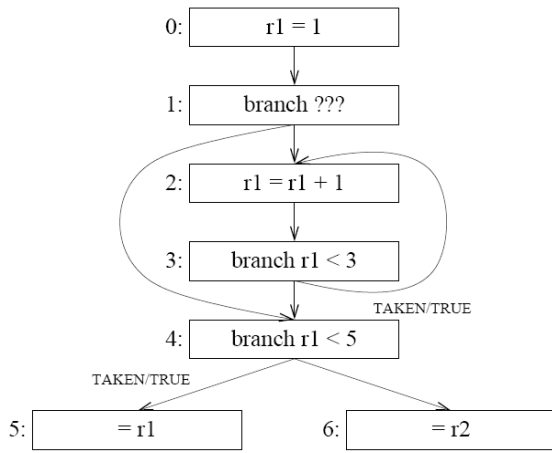
- Register r2 looks live by our live variable analysis, but is *not*.
- In general, today's compilers do not determine exactly how execution will proceed.
- Inaccuracy must lead to *conservative approximations*.
 - Optimizations must only be applied when proven safe.
 - Conservatism may not always compute best results.
- *MCI in ML* uses the terms *statically live* and *dynamically live*.

Conservative Approximations Example

Register Allocation:



New Dataflow Analysis



Limitation of Dataflow Analysis

